Midterm Questions Study Guide (Draft!)

1 [binary search tree]: write a function that returns the minimum and maximum lengths of a simple path from root to each leaf in a binary search tree. You may add add additional helper functions if you require them. Assume the following declaration:

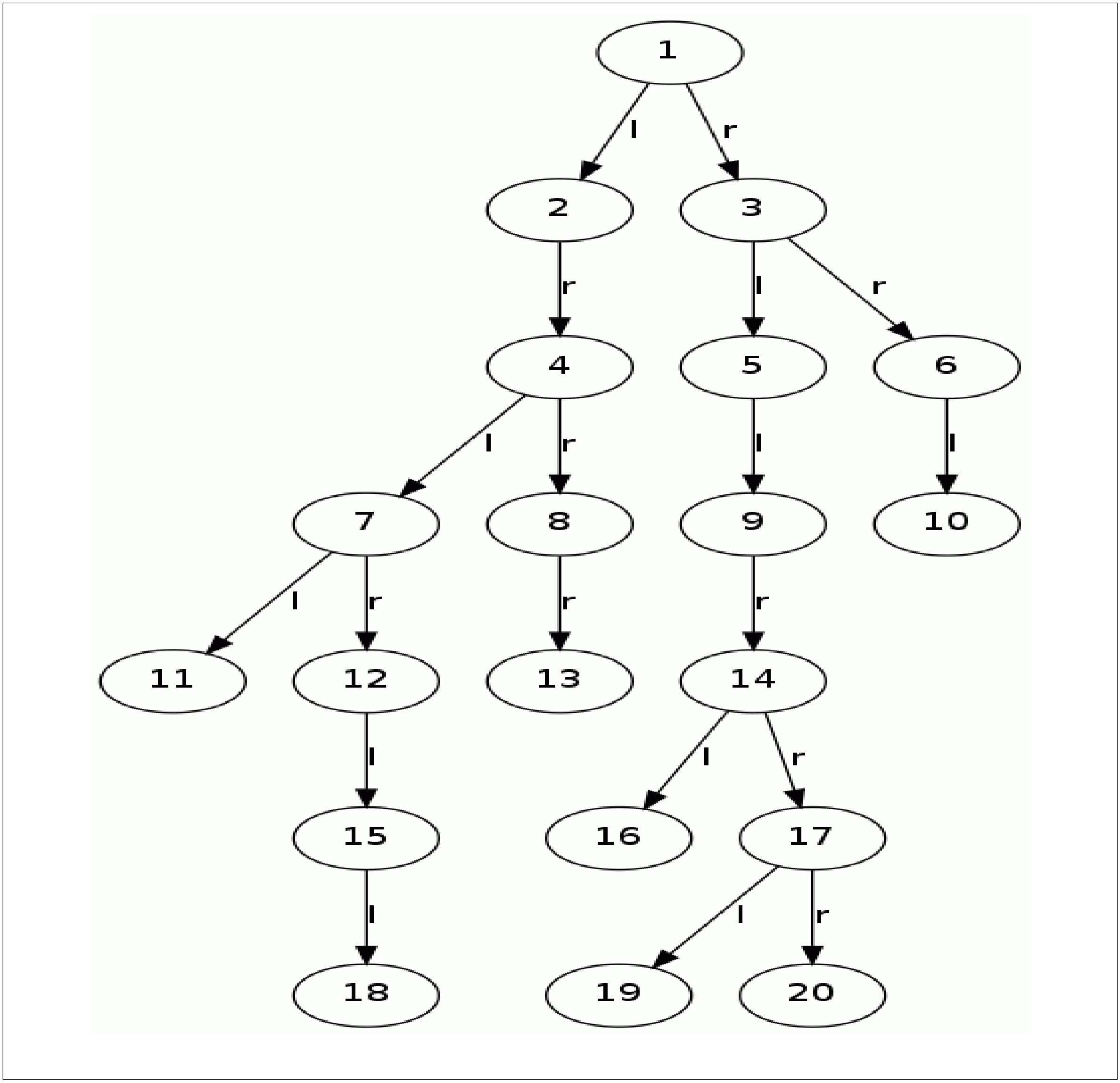
```
class BST {
  public:
     // your function here
  private:
    struct Node {
      static const int RED = 0;
      static const int BLACK = 1;
      int color;
      SomeType* Data;
      Node* left;
      Node* right;
    };
    Node* root;
};
```

2 [binary search tree (this will be overkill compared to any question that makes the midterm, but if you can dash it off easily you should have no trouble with the exam question)]: Assume the above declaration. Write a function to verify that a BST has the following red-black properties (returns true if valid, false otherwise):

1. root is black
2. (synthetic) leaves are black
3. children of a red node are black
4. interior nodes have 2 children (which may include synthetic black leaves, you don't care whether nodes are sythetic or not)
5. every simple path from the root to a leaf have the same black height [the red-black condition is stronger, but we'll go with this]

[Fair game for the test might be any single one of the above, or similar conditions for a 2-3 Tree or AVL]

3 [binary tree (the exam question will have fewer nodes!)]: Given thie following tree, decorate the nodes with pre- and post-order enumerations:



4 [binary search tree] Assuming the tree above is a binary search tree, what is the immediate predecessor and successsor of node 4? Node 9? Node 20?

5 [pointer arithmetic]: Consider will the following program.   Fill in the blanks in the output

```
#include <iostream>
#include <iostream>
using namespace std;

typedef unsigned long UL;

struct Thing1 {
  char a;
  short b;
  int c;
  double d;
};

int main() {
  Thing1 a[8];
  Thing1 *p = &a[2];
  Thing1 *q = &a[5];

  cout << "size of Thing1:  " << sizeof(Thing1) << endl;
  cout << "size of p:       " << sizeof(p) << endl;
  cout << "size of a:       " << sizeof(a) << endl;
  cout << "size of a[3]     " << sizeof(a[3]) << endl;
  cout << "sizeof &a[3]     " << sizeof(&a[3]) << endl;

  cout << "p - a:           " << p - a << endl;
  cout << "q - p:           " << q - p << endl;
  cout << "UL(p) - UL(a):   " << UL(p) - UL(a) << endl;
  cout << "UL(q) - UL(p):   " << UL(q) - UL(p) << endl;

  cout << "UL(a)            " << UL(a) << endl;
  cout << "UL(&a[0])        " << UL(&a[0]) << endl;
  cout << "UL(p)            " << UL(p) << endl;

  return 0;
}
```

OUTPUT:

```
size of Thing1:  16
size of p:       8
size of a:       _______________
size of a[3]     _______________
sizeof &a[3]     _______________
p - a:           _______________
q - p:           _______________
UL(p) - UL(a):   _______________
UL(q) - UL(p):   _______________
UL(a)            140734332027424
UL(&a[0])        _______________
UL(p)            ________________
```

5 [pointer arithmetic]: Consider will the following program.   Fill in the blanks in the output.

```
#include <iostream>
using namespace std;

typedef unsigned long UL;
```

```
struct Thing1 {
  char a;
  short b;
  int c;
  double d;
};

int main() {
  Thing1 my_thing;

  cout << "size of my_thing:   " << sizeof(my_thing) << endl;
  cout << "size of my_thing.a: " << sizeof(my_thing.a) << endl;
  cout << "size of my_thing.b: " << sizeof(my_thing.b) << endl;
  cout << "size of my_thing.c: " << sizeof(my_thing.c) << endl;
  cout << "size of my_thing.d: " << sizeof(my_thing.d) << endl;
  cout << "a + b + c + d:      " << sizeof(my_thing.a) +
sizeof(my_thing.b) + sizeof(my_thing.c) + sizeof(my_thing.d) << endl;

  cout << "UL(&my_thing):      " << UL(&my_thing) << endl;
  cout << "UL(&my_thing.a):    " << UL(&my_thing.a) << endl;
  cout << "UL(&my_thing.b):    " << UL(&my_thing.b) << endl;
  cout << "UL(&my_thing.c):    " << UL(&my_thing.c) << endl;
  cout << "UL(&my_thing.d):    " << UL(&my_thing.d) << endl;

  return 0;
}
```

OUTPUT:

```
size of my_thing:   16
size of my_thing.a: 1
size of my_thing.b: 2
size of my_thing.c: 4
size of my_thing.d: 8
a + b + c + d:      15
UL(&my_thing):      140735184671600
UL(&my_thing.a):    _______________
UL(&my_thing.b):    _______________
UL(&my_thing.c):    _______________
UL(&my_thing.d):    _______________
```

6 [pointer arithmetic]: Given the output above, why the difference between sizeof(my_thing) and the sum of the sizeof of each field?

7 [pointer arithmetic]: why are sizeof(Thing1) and sizeof(Thing2) different?

```
#include <iostream>
```

```
using namespace std;

struct Thing1 {
  char a;
  short b;
  int c;
  double d;
};

struct Thing2 {
  short b;
  int c;
  char a;
  double d;
};

int main() {
  cout << "sizeof(char): " << sizeof(char) << endl;
  cout << "sizeof(short): " << sizeof(short) << endl;
  cout << "sizeof(int): " << sizeof(int) << endl;
  cout << "sizeof(long): " << sizeof(long) << endl;
  cout << "sum of sizes: " << sizeof(char) + sizeof(short) +
sizeof(int) + sizeof(long) << endl;

  cout << "sizeof(Thing1): " << sizeof(Thing1) << endl;
  cout << "sizeof(Thing2): " << sizeof(Thing2) << endl;

  return 0;
}
```

OUTPUT:

```
sizeof(char): 1
sizeof(short): 2
sizeof(int): 4
sizeof(long): 8
sum of sizes: 15
sizeof(Thing1): 16
sizeof(Thing2): 24
```

8 [2-3 tree]: given the following trace output, draw a bubbles-and-arrows diagram of the 2-3 tree:

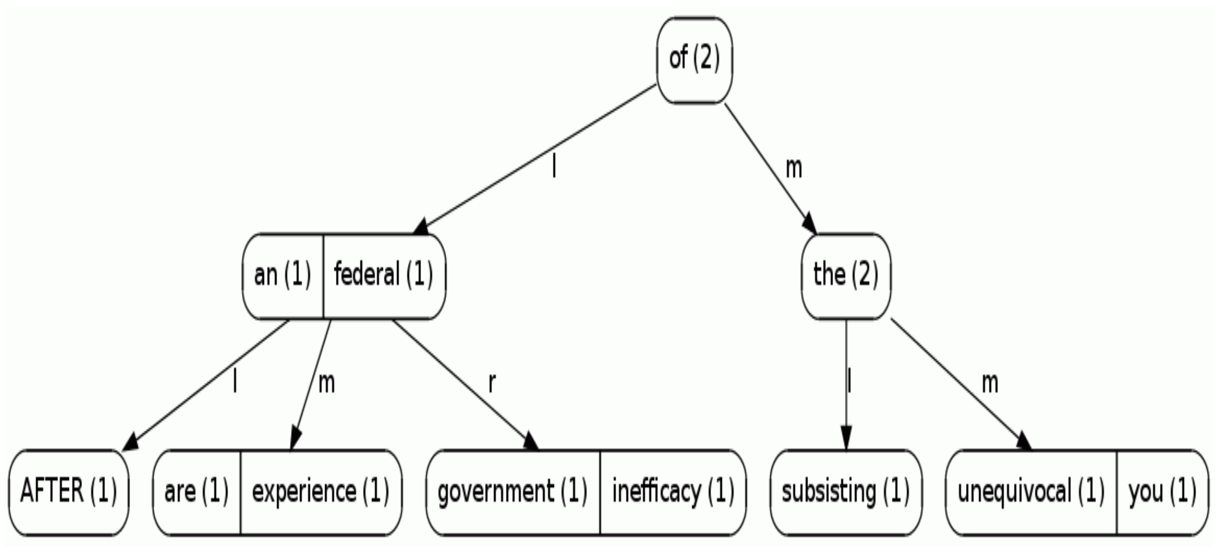
```
0x16fb280 (0x16fb100, 0x16fb250, 0)
   lazy
0x16fb100 (0x16fb040, 0x16fb190, 0)
   fox
0x16fb040 (0, 0, 0)
   brown
   dog
0x16fb190 (0, 0, 0)
   jumps
0x16fb250 (0x16fb220, 0x16fb0d0, 0)
   quick
0x16fb220 (0, 0, 0)
   over
0x16fb0d0 (0, 0, 0)
   the
```

9 [2-3 tree]: Given the output above, assume p is a pointer to a tree node containing the word “quick”, what is the value of the pointer p->middle?

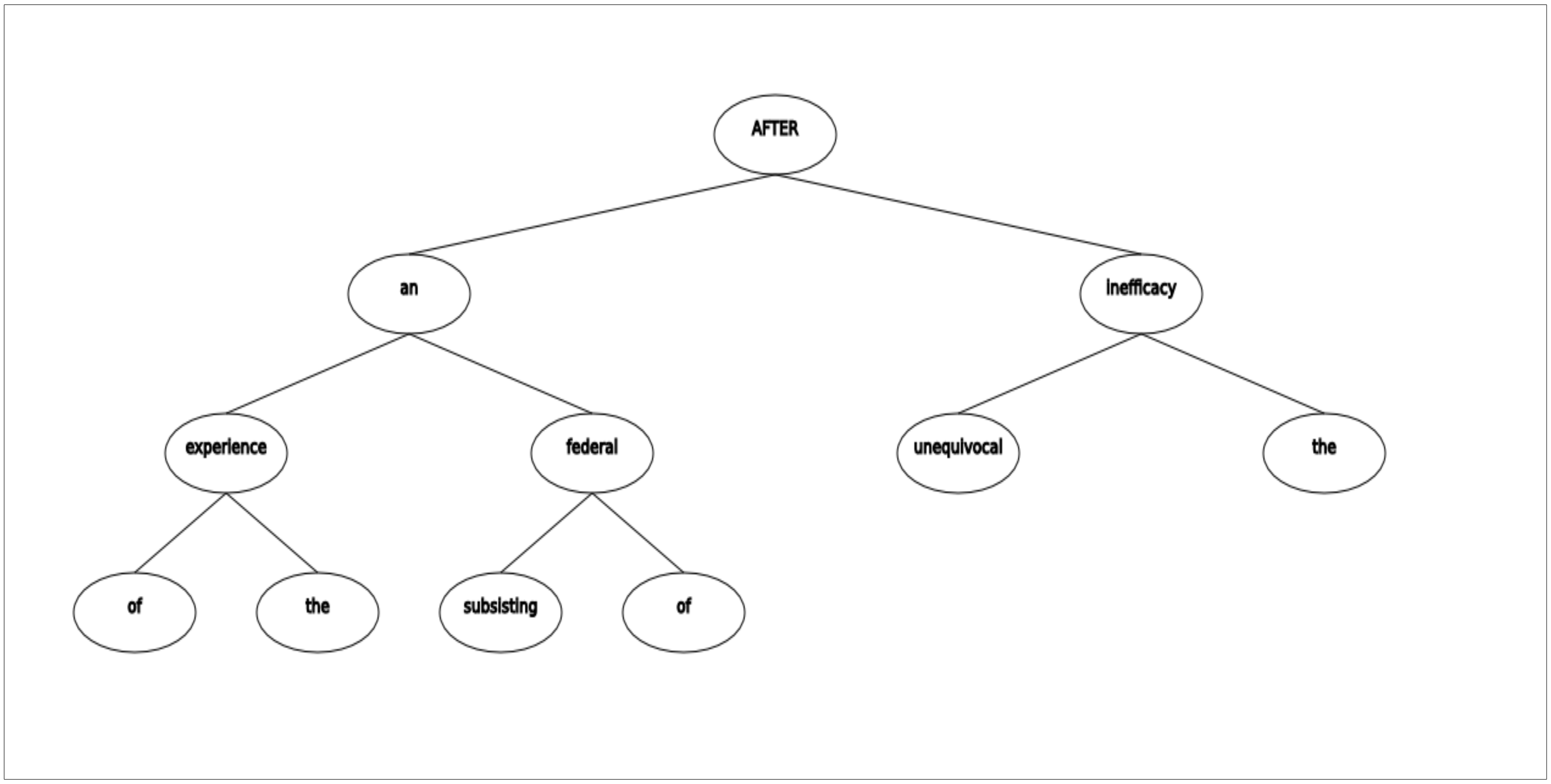
10 [2-3 tree]: Given the following declaration, write a function that calculates the percentage of unused key elements:

```
class Btree {
  public:
    // your function here
  private:
    class Node {
      Data* key1;
      Data* key2;
      Node* left;
      Node* middle;
      Node* right;
    };
    Node* root;
};
```

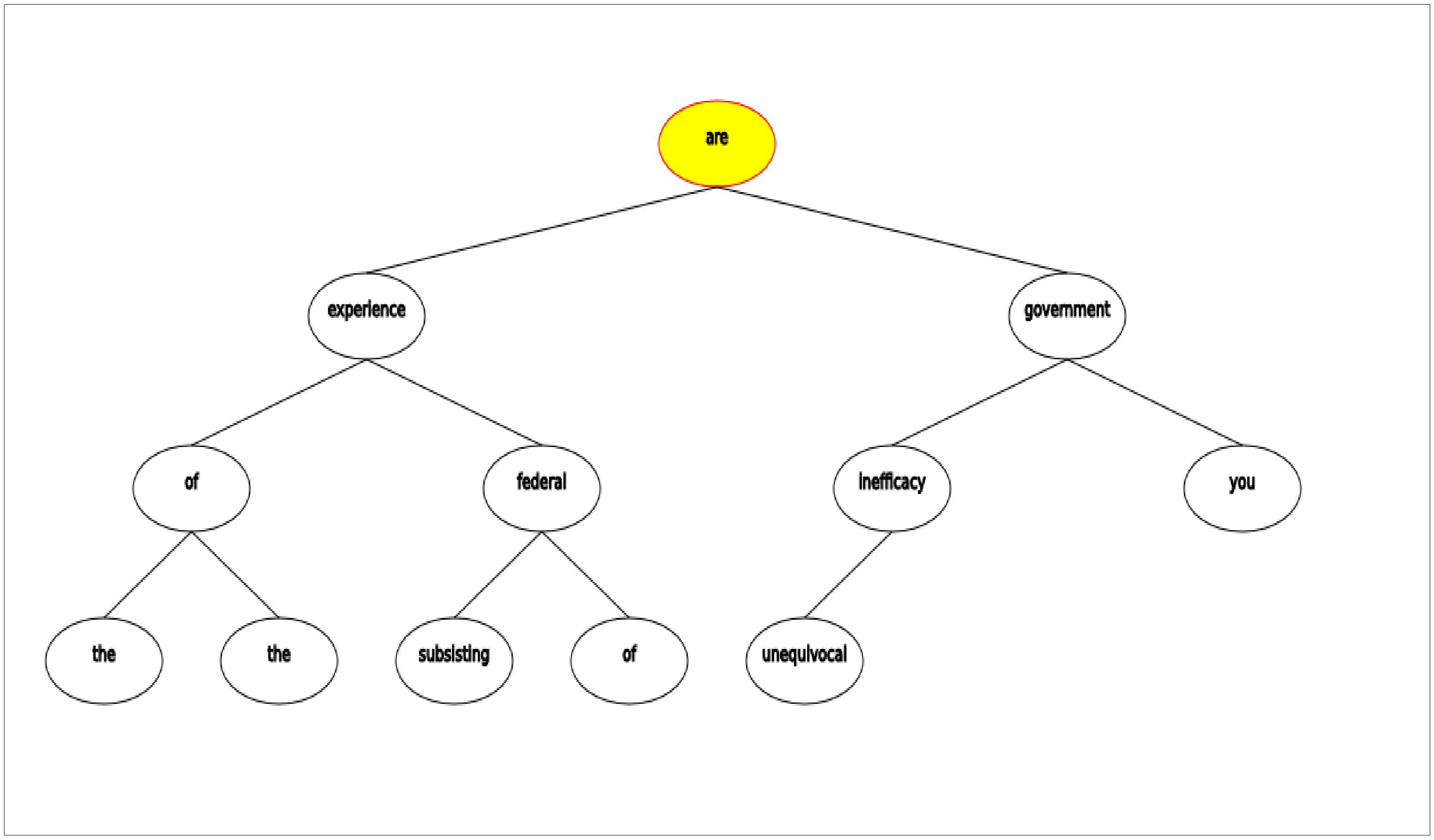
11 [2-3 tree]: Redraw the following tree after inserting the word “called”:



12 [priority queue]: redraw the following heap tree structure after inserting the word “government”:



13 [priority queue]: redraw the following heap tree after removing the minimum element:


are
experience
government
of
federal
inefficacy
you
the
the
subsisting
of
unequivocal